3 Командная строка и командный интерпретатор (shell)

3.1 Канонический ввод, командная строка

В типичном случае события ***консоли*** (нажатия отдельных клавиш) обрабатываются процессом-демоном, ответственным за обслуживание консольного ввода, который преобразует при необходимости символы и формирует из них завершенные ***строки***.

Строки передаются (с буферизацией) в ***поток*** – обычно стандартный поток ввода (**`stdin`**) активного процесса этой консоли. Признак завершения строки – символ «***конца строки***» (***End-of-line***, ***EOL***), обычно в это символ «***новой строки***» (***New line***, ***NL***), обычно записываемый как «**`\n`**».

Ввод осуществляется с «эхом» на консоль – с отображением вводимых символов (за это отвечает процесс-демон, отсылающий на консоль только что полученные символы). Однако процесс, ожидающий ввода, получает данные в виде уже законченный и «подтвержденных» строк.

Такой способ организации ввода называют ***каноническим***. Им пользуется большинство обычных прикладных программ.

Рис. – канонический ввод

Поскольку «первичный» ввод строки, ее отображение и редактирование осуществляются фактически системным процессом, ее формат, алфавит, правила редактирования и т.д. едины для всех прикладных программ с текстовым режимом ввода-вывода. Эти же правила (в основном) распространяются и на ввод программ с графическим интерфейсом, и на работу со строками вообще.

Некоторые интерактивные программы (например, по очевидным причинам **`passwd`**) не пользуются каноническим вводом, запрашивая данные на более низком уровне.

Строка – последовательность символов (обычных и специальных), завершающаяся специальным символом «конца строки».

Алфавит – множество допустимых символов

Символы – обычные печатные: буквы, цифры, пунктуационные и некоторые другие знаки (если им не придается специальный смысл)

Специальные символы (***метасимволы***) – символы, имеющие специальный смысл и, в общем случае, не имеющие адекватного печатного изображения.

Восприятие символа как обычного или специального зависит от конкретной программы. Часть специальных символов распознаются и применяются на уровне системы (например, демонами консоли), другие – лишь конкретными драйверами, демонами, прикладными программами.

Множества специальных символов различных программ не совпадают друг с другом, но часто пересекаются – имеют общие подмножества. Это может приводить к коллизиям, требующих особых мер для их разрешения.

Экранирование – отмена специального смысла отдельных символов или подстрок с помощью других специальных символов-***экранов***. Типично использование в качестве экрана «обратного слэша» – «\», также часто экранирующие свойства имеют кавычки.

Экранирование одиночного специального символа:

спец_символ

Например, экранирование самого символа '\':

**** → ****

Экранирование конца строки:

Часть строки
Продолжение строки → **Часть строкиПродолжение строки**

Символ экранирования «****» работает как переключатель – он также придает специальный смысл обычным символам, например: «**\b**», «**\n**», «**\r**», «**\t**» и так далее.

Часть строки\nПродолжение → **Часть строки**
Продолжение

Экранирование строк – подавление специального смысла символов, включая разделители:

"*экранируемая_строка*"
'*экранируемая_строка*'

Обратные кавычки «`…`» – не экран, а обращение к команде (см. далее)

Командная строка – подход к организации интерфейса пользователя в интерактивном (диалоговом) режиме: ввод команд пользователя посредством консоли и получение результатов на консоль.

В силу особенностей Unix-систем консольный ввод-вывод легко подменяется чтением-записью файлов, что позволяет с минимальными затратами перейти от диалогового режима к пакетному: к выполнению не отдельных команд, а файлов с последовательностью команд (***сценариев*** или ***скриптов***).

Первая же из программ, реализующих интерфейс командной строки – командный интерпретатор или оболочка (shell).

3.2 Командный интерпретатор – место, роль, функции

Командный интерпретатор или ***оболочка*** (***Shell***) – программа, обеспечивающая взаимодействие с пользователем посредством интерфейса командной строки. В соответствии с командами пользователя или с интерпретируемым «сценарием» оболочка обеспечивает исполнение других программ (процессов) в нужном порядке и с нужным взаимодействием между собой. Таким образом, оболочка занимает место посредника между пользователем и системой (и другими программами).

Особое положение shell обусловлено также тем, что он часто является ***лидером сеанса*** пользователя.

Некоторые (очень немногие из всего множества!) виды командных интерпретаторов:

`sh` – ***Bourne shell***, «классический» интерпретатор Борна (Stephen Bourne) – создан в 1979 г. как замена более раннего интерпретатора Томпсона, считается стандартным и присутствующим «по умолчанию» в большинстве Unix-систем. Ожидаемо отстает по возможностям от более современных, но некоторые скрипты (особенно для запуска системы и администрирования) часто пишутся в расчете на **`sh`**.

`bash` – «***Bourne again shell***», обновленный (1989 г., Brian Fox) вариант **`sh`**, сохраняющий с ним максимальную совместимость, но дополненный многими полезными возможностями.
В качестве основного shell Linux-систем чаще всего встречается **`bash`**.

csh – «***C-shell***», появился как альтернатива **sh** в BSD-системах и часто используется в них как основной. Синтаксис приближен к языку C (очень условно приближен), есть встроенная поддержка арифметики с числами, имеется много отличий от **sh**. В сочетании с заметными отличиями также и BSD-версий команд (утилит) это создает серьезные проблемы при переносе скриптов (на практике часто проще обеспечить наличие нужного интерпретатора).

ksh – «***Korn shell***», интерпретатор Корна (David Korn), значительное расширение возможностей по сравнению с **sh** (арифметика с числами, массивы, и т.д.), но по синтаксису уходил от него не так далеко, как **csh**. Нововведения в **ksh** частично использованы в **bash**.

zsh – «***Z shell***», надстройка над **bash** («is built on top of bash»), считается основным в macOS и Kali Linux.

И другие. При желании можно написать собственный.
(А главное, попытаться сделать его популярным.)

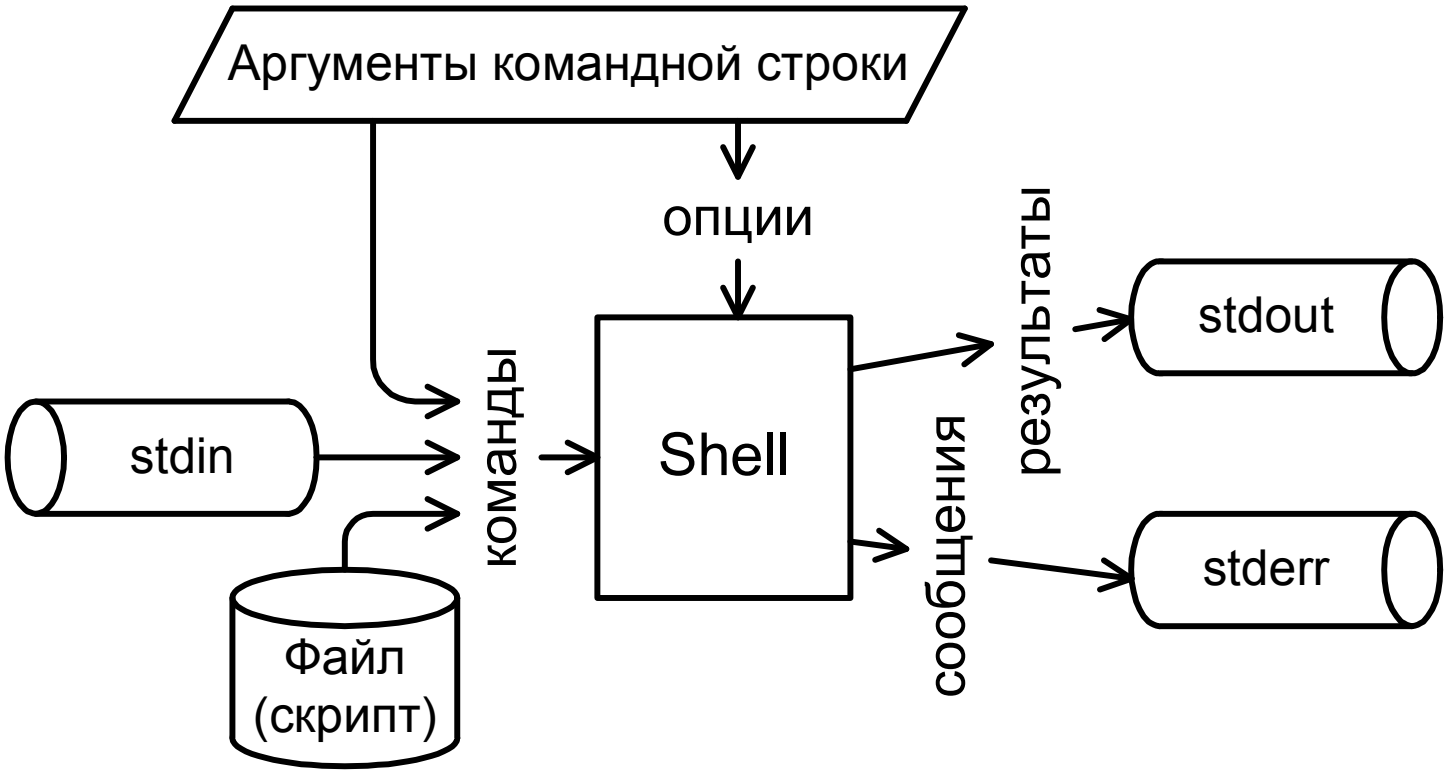
3.3 Командная строка, команды, сценарии

Несмотря на различия, все командные интерпретаторы придерживаются одних и тех же принципов функционирования.

Режимы выполнения shell:

– ***интерактивный*** – последовательное выполнение команд, вводимых пользователем

– ***пакетный*** (***batch***) – выполнение интерпретируемого сценария (***скрипта***)



Выполнение командного интерпретатора

Способы выполнения shell (в качестве примера пусть будет **sh**):

Новая копия интерпретатора в интерактивном режиме:

sh

Новая копия интерпретатора для выполнения команд(ы) из одной строки:

sh -c *string_with_command*

Попытка выполнить скрипт как команду (неявный запуск новой копии интерпретатора), необходимы права на выполнение файла ***myscript.sh***:

./myscript.sh

То же в текущем экземпляре интерпретатора:

. *./myscript.sh*

Выполнение скрипта явным образом в новой копии интерпретатора, для файла ***myscript.sh*** достаточно права на чтение:

```
sh myscript.sh
```

То же, но имя скрипта для интерпретатора останется не известно (перенаправление ввода):

```
sh < myscript.sh
```

То же, но сложнее (демонстрация конвейеризации команд):

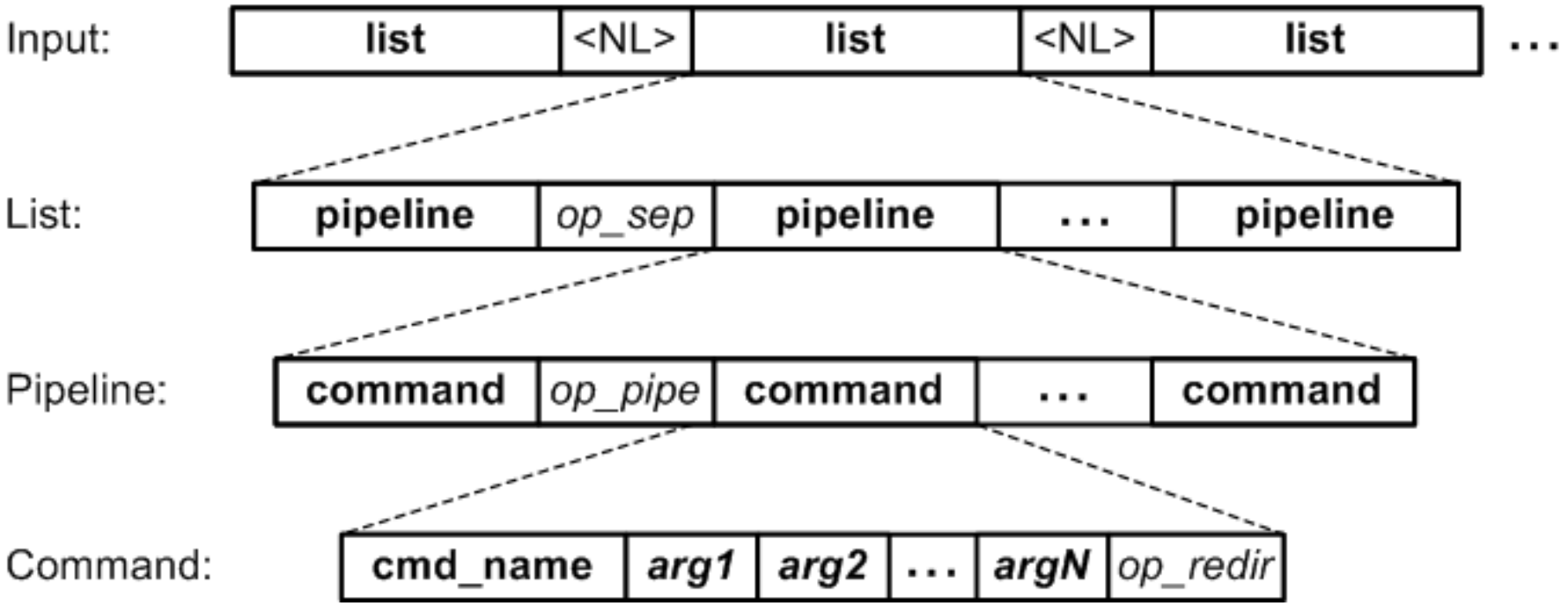
```
cat myscript.sh | sh
```

То же, но скрипт генерируется динамически (потенциально сильно, но сложно и обычно непрактично, используется редко):

```
./myscriptgenerator | sh
```

(Варианты с перенаправлением вывода для упрощения не рассматриваются)

В любом случае командный интерпретатор получает на вход поток строк, которые он интерпретирует как команды и пытается их выполнить.



Входной поток команд

Структура строки (до общего «конца строки»):

– «***список***» команд (***list***) – один или несколько «***конвейеров***», разделенных операторами-***разделителями*** ***`op_sep`***: «`;`»,«`&&`» или «`||`», может завершаться операторами «`;`» или «`&`»
Кодом завершения списка становится код завершения последнего его конвейера.

– «***конвейер***» (***pipeline***) – одна или несколько одиночных ***команд***, разделенных «***операторами управления***» (***control operator***) ***`op_pipe`***: «`|`» или «`|&`», а также &&, ||, ...
Кодом завершения конвейера становится код завершения последней его команды.

– одиночная ***команда*** (***simple command***) – имя команды, возможно сопровождаемое ***аргументами*** и операторами ***перенаправления*** ***`op_redir`***

Общий вид одиночной команды:

команда **[** ***аргумент1 аргумент2*** **…** **]**

Команды делятся на внешние и внутренние:

– ***внешние*** – исполняемые файлы (любые), доступные в файловой системе

– ***внутренние*** – выполняются непосредственно самим интерпретатором shell.

Аргументы команд:
– ***опции*** (начинаются с символов «дефис», «двойной дефис»
– прочие ***параметры*** (например, список имен файлов)

Обычно опции и другие параметры допустимо чередовать, но часто стараются вынести все опции в начало, после чего остается список из произвольного числа параметров.

После опции может отдельно идти ее значение:

gcc -o *myexecfile*

«Однобуквенные» опции можно объединять:

ls -l -d * → **ls -ld ***

Таким образом:

**команда [*-opt1 -opt2 --options3*] **
[*парам1 парам2* …] [*op-redir*]

Наборы внутренних команд и их синтаксис специфичны для конкретных интерпретаторов, частично пересекаются. Определены «стандартный» набор внешних команд и их синтаксис, специфичные для семейств, подсемейств и т.д. ОС.

Linux – стремление объединить стандарты различных систем, в том числе поддержка альтернативных версии синтаксиса.

Составная команда (***compound command***) – список команд, сгруппированный с помощью скобок, или конструкция цикла или ветвления (см. ниже).

Составная команда, выполняемая в отдельном окружении (в новой копии shell):

```
( список команд )
```

Составная команда, выполняемая в текущем окружении (разделитель «*;*» или NL):

```
{ список команд ; }
```

Более подробное рассмотрение синтаксиса shell (в первую очередь **sh** и **bash**), конструкций, а также отдельных команд – в следующем разделе.